

An Extended RBAC Profile of XACML

Diala Abi Haidar^{1,2}, Nora Cuppens-Boulahia¹, Frederic Cuppens¹, Herve Debar²

¹ ENST Bretagne, 2 rue de la Châtaigneraie, 35512 Cesson-Sévigné Cedex, France

² France Telecom R&D Caen, 42 rue des Coutures BP 6243, 14066 Caen, France

ABSTRACT

Nowadays many organizations use security policies to control access to sensitive resources. Moreover, exchanging or sharing services and resources is essential for these organizations to achieve their business objectives. Since the eXtensible Access Control Markup Language (XACML) was standardized by the OASIS community, it has been widely deployed, making it easier to interoperate with other applications using the same standard language. The OASIS has defined an RBAC profile of XACML that illustrates how organizations that would like to use the RBAC model can express their access control policy within this standard language. This work analyzes the RBAC profile of XACML, showing its limitations to respond to all the requirements for access control. We then suggest adding some functionalities within an extended RBAC profile of XACML. This new profile is expected to respond to more advanced access control requirements such as user-user delegation, access elements abstractions and contextual applicability of the policies.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*Access controls*; D.2.8 [Software Engineering]: Software Architectures—*Languages, Patterns*; D.3.2 [Programming Languages]: Language Classifications—*Extensible languages*

General Terms

Security, Languages

Keywords

Access control, XACML, RBAC, OrBAC

1. INTRODUCTION

Security for web services has already attracted attention [10, 8, 18, 11]. Since web services are based on distributed

and heterogeneous systems, there is a need for platform-independent, generic and lightweight communication mechanisms to encourage their interoperability. This is why XML stands out as a natural answer for web services languages. It has also become the natural choice for a common security policy language due to the ease with which its syntax and semantics can be extended.

One of these security policy languages is the eXtensible Access Control Markup Language (XACML), an OASIS standard [4]. XACML describes both a policy language and an access control decision request/response language. The policy language is used to describe general access control requirements to resources in the information system. The request/response language allows us to form a query to ask whether or not a given action should be allowed and the response will convey the answer for this query. The answer should contain one of these four values: Permit (access allowed), Deny (access denied), Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (this service has no policies that apply to this request).

Other aspects in the access control domain may require more of a language. This is why OASIS has also defined profiles for XACML that address other requirements in this field. The SAML profile of XACML [5] is suitable for supporting the assertion and protocol mechanisms needed by XACML. The Security Assertion Markup Language (SAML) [6, 7], as its name suggests, provides means to write assertions regarding the identity, attributes, and entitlements of a subject, and defines a protocol for exchanging these assertions between entities. The core and hierarchical role based access control (RBAC) profile of XACML [3] defines a profile for the use of XACML to meet the requirements of RBAC [16, 25]. XACML is a powerful, standard language that may offer possibilities to express other requirements. This is what we have intended to study in this paper.

The paper is organized as follows. Section 2 introduces the RBAC profile of XACML. Section 3 explains the needs to have an extended profile by specifying the limits of this RBAC profile and analyzing additional requirements. Section 4 presents our proposition for an extended profile that is adaptable enough to express a large class of existing access control models. Section 5 illustrates the implementation of our profile and its application. Finally section 6 concludes the paper and gives the perspectives to this work.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CCS'06, October 30–November 3, 2006, Alexandria, Virginia, USA.

Copyright 2006 ACM 1-59593-518-5/06/0010 ...\$5.00.

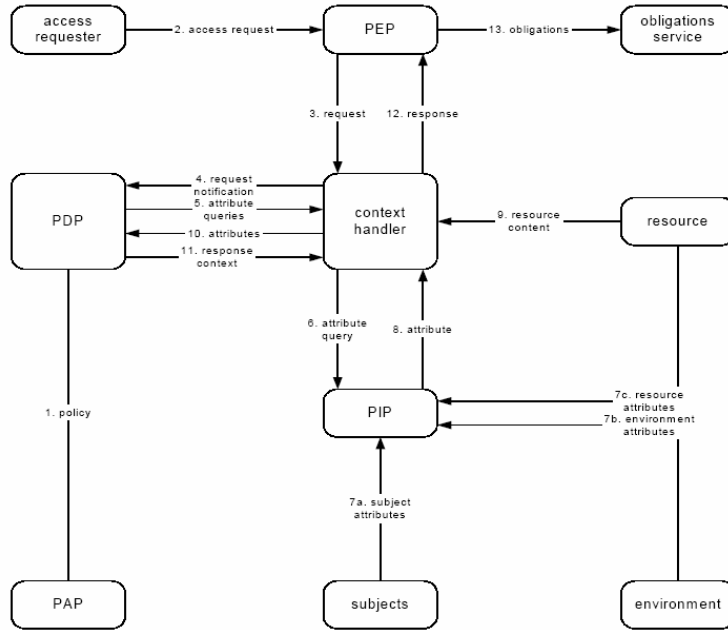


Figure 1: The OASIS XACML non-normative data-flow diagram

2. RBAC PROFILE OF XACML

2.1 XACML in brief

Before going further in this discussion, let us go back to XACML to examine its characteristics. We have already introduced XACML as a policy language and an access control decision request/response language (both written in XML). Regarding the components [4], at the root of all XACML policies is a Policy or a PolicySet. A PolicySet is a container that can hold other Policies or PolicySets, as well as references to remote policies. A Policy represents a single access control policy, expressed through a set of Rules. Each XACML policy document contains exactly one Policy or PolicySet root XML tag. A Policy consists of a Target, one or more Rules, and an optional set of Obligations. Target specifies a set of Subjects, Resources, Actions and Environments. It uses boolean evaluations to determine if the values in a request meet the conditions of the Target. If all of the conditions are met, the Target's associated Policy, PolicySet, or Rule applies to the request.

PolicySet and Policy may contain resp. multiple policies and rules, each of which may evaluate to different access control decisions. For this purpose, XACML defines a collection of Combining Algorithms. *Policy Combining Algorithms* are used by the PolicySet to reconcile the decisions each policy makes and *Rule Combining Algorithms* are used by the Policy for reconciling the decisions each rule makes. Finally, obligations may be added in a Policy or PolicySet. Obligations are a set of operations that must be fulfilled in conjunction with an authorization decision (permit or deny authorization decision).

The OASIS standard defines a data-flow diagram (see the figure 1).

The typical setup is that someone wants to perform some

action on a resource. She will make a request (2) to whatever actually protects that resource (like a filesystem or a web server), which is called a Policy Enforcement Point (PEP). The PEP will form a request, in its native request format, based on the requester's attributes, the resource in question, the action and other information pertaining to the request. This request is sent to the context handler (3) that constructs an XACML request context for a Policy Decision Point (PDP) (4), which will look at the request and some policy that applies to the request. The policies have been written by the Policy Administration Point (PAP) and made available to the PDP (1). Sometimes, the PDP may require additional attributes while evaluating the request. In this case attribute queries are sent to the context handler (5), this latter requests the attributes from the Policy Information Point (PIP) (6) then (7, 8, 9) sends them back to the PDP (10). The PDP will finally evaluate the policy and come up with an answer about whether access should be granted (11). That answer is returned to the PEP, via the context handler that translates it to the native response format of the PEP (12). The PEP can then allow or deny access to the requester and possibly fulfills some obligations (13).

2.2 RBAC

A standard language to express the access control is essential but it is just a way to write policies and rules. It is important to have a model that a security manager can use to express all the security requirements. The ultimate goal is to be able to express such a model using the existing languages. There are many access control models in the literature (TBAC [26], DAC [19], OrBAC [22], ABAC [27]...), the most frequently cited being the Role Based Access Control (RBAC) [25].

The basic concept of the RBAC model is that users are

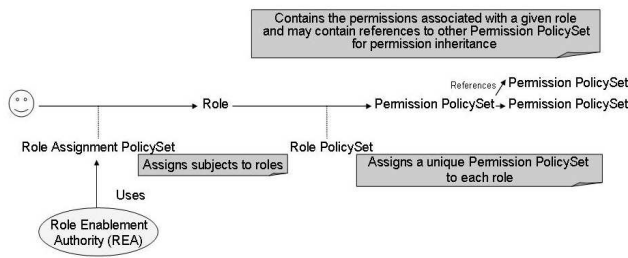


Figure 2: The summary of RBAC profile principles

assigned to roles, permissions are assigned to roles and users acquire permissions by being members of roles. The user-role assignment can be a many-to-many relation in the sense that a user can be assigned to many roles and a role can have many users. Similarly, the permission-role assignment is also a many-to-many relation. The RBAC model is organized in four levels [24] each including the requirements of the basic RBAC: the flat (or core) RBAC, the hierarchical RBAC that adds requirements for supporting role hierarchies and the constrained RBAC that adds constraints on the hierarchical RBAC. The constraints may be associated with the user-role assignment (for static separation of duty) or with the activation of roles within user sessions (for dynamic separation of duty). The last level is the symmetric RBAC (also called consolidated) that adds a requirement for permission-role review. This is essential in any authorization management to identify and review the permissions assignment, i.e. the relation between permissions and roles.

The main benefit of this model is the ease of administration of security policies and its scalability. When a user moves inside an organization and has another function, the only thing the administrator needs to do is to revoke the existing user-role assignment and assign her a new role. There is no need to revoke the authorizations she had before and she will be granted new authorizations assigned to her new role. Adding to that, the role hierarchy defined in this model, where a given role can include all the permissions of another role, is a way of having a well structured access control that is the mirror of the organization structure. Finally the RBAC model supports the delegation of access permissions between roles. A role can delegate its role or part of its role to another role [12].

2.3 The core and hierarchical RBAC profile of XACML

Using existing access control models eases access control management. The remaining question is how we can use this model within these existing access control languages. The RBAC profile of XACML expresses a way to use the standard XACML access language within the RBAC model. Figure 2 summarizes the principal PolicySets defined in this profile.

In this profile, the *Role Enablement Authority* (REA) is in charge of the assignment of subjects to roles. This entity may use the Role Assignment PolicySet that defines which roles can be assigned to which subjects. The REA may maintain a list of all the roles that are defined in the organization and when asked about the role that can be as-

signed to a subject, this entity makes a request against the *Role Assignment Policies* (RAP) for each candidate role. The holders of a given role are associated with a Permission PolicySet through the Role PolicySet that must reference a single corresponding Permission PolicySet. This latter contains the permissions associated with a given role and may contain references to other Permission PolicySets. In this way a role can inherit all the permissions associated with the role of the referenced Permission PolicySet. In this manner the role hierarchy is implemented.

The XACML policies using this profile can also express conditions on the application of particular permissions through the XML element `<Condition>`. This is why the profile extends the RBAC requirements by providing a means of limiting permissions associated to a given role to a given condition such as time period of the day.

3. THE NEED FOR AN EXTENDED PROFILE

3.1 The limits of the RBAC profile of XACML

The OASIS standard for the RBAC profile of XACML [3] takes into consideration the concept of role in the expression of access control policies. Unfortunately, it only corresponds to the core and hierarchical RBAC profile of XACML. In other words it is not sufficient to express the constrained RBAC. Moreover, the roles are considered to be already assigned when the authorization request is submitted for evaluation. This is why static and dynamic separation of duty is not treated in this profile.

Another limit of this profile is the way it implements the hierarchy. It considers that the *PolicySetIdReference* or the *PolicyIdReference* in a role r permission's *PolicySet* are used to reference the permissions that are related to another role from which the role r inherits the authorizations. A subject inherits via this process all the permissions of the referenced role. This mechanism for implementing the hierarchy makes it impossible to implement a delegation mechanism by using references to the permissions of the delegated role. Considering that a physician x decided to go on vacation and delegate her permissions to her assistant y . Each of the roles *assistant* and *physician* is associated with a corresponding Permission PolicySet through the Role PolicySet. In the case of hierarchy between these roles, if the assistant was a senior role of the role *physician*, it was easy to reference the Permission PolicySet of the *physician* in the Permission PolicySet of the role *assistant*. Then allowing all the subjects who play the role *assistant* to have the permissions of the *physician* role. Whenever we need to implement the delegation between the roles using the same reference process - referencing the delegated Permission PolicySet (physician one) in the Permission PolicySet of the assistant role - all the persons assigned to the role *assistant* inherit the permissions of the *physician* role.

In the case of partial delegation, we may not need to give the delegated permissions to all the persons who are assigned to a role. In our example, we need that only the corresponding assistant, the subject y , inherits all the permissions of physician x . Then the physician x must assign her assistant y to the role *physician* during her vacation period. Now, considering that the physician wanted to delegate only a part of her permissions to her assistant, it becomes necessary to create a new role, *delegatedPhysician*, that has another

Permission PolicySet which contains the selected delegated permissions. The assistant is only assigned to the new `delegatedPhysician` role. We may also need to use the condition element in XACML in the Role Assignment PolicySet for the assignment of the new role to the assistant. This is to specify that this role is assigned to the assistant only during the physician's vacation otherwise the assistant may maintain the permissions even after the return of the physician x .

This case of partial delegation is too complicated to manage in the RBAC profile of XACML. Not only should we create a new role but also a new Permission PolicySet. Furthermore, we should make changes in the Role PolicySet to associate the new role with its new Permission PolicySet.

3.2 The architecture

Even though the RBAC profile defines the REA and the different new PolicySets that should be created to express the requirements of the RBAC model, it does not explicitly define or implement the data-flow in a precise architecture. There may be different ways to implement these new components. We will then propose our implementation, describing the place that the REA will take into the data-flow diagram. We will define the other entities that should be created to implement the extended functionalities of the existent profile to go beyond its limitations.

3.3 A better use of the XACML functionalities

3.3.1 Prohibition

Several models include prohibitions or negative authorizations when specifying an access control policy [15, 13, 20]. A negative authorization policy specifies actions that subjects are forbidden to perform. RBAC does not allow prohibitions, whereas they are included in the XACML policies through the effect of a `deny` rule. Prohibition is useful for example whenever we need to limit the propagation of permissions in hierarchies. Prohibitions can also be used as exceptions to remove access rights from subjects if the need arises. Many other situations make these negative authorizations very useful especially in the case of decentralized policy management where each administrator does not have a vision of the whole policy. Therefore, we suggest using XACML negative authorizations to extend the RBAC model.

3.3.2 Context

XACML policies can express conditions on the application of particular permissions through the XML element `<Condition>`. Nevertheless, this is not sufficient to include all the contextual conditions. The `<Condition>` element is a boolean function over subject, resource, action and environment attributes of the request. It can be used to express conditions such as: (1) a certain period of the day for example from 9am till 7pm which can be seen as an environment attribute or, (2) the designated doctor of a patient which is deduced from the subject and resource attributes.

We may need to express conditions where the information to be evaluated is not carried in the request. Then other requests are generated by the information system itself to find missing information. For instance, whenever the access to a resource is limited to a number of times for a given subject, we need to get the history of the subject accesses to evaluate the number of times she had an access to this resource.

Therefore, we need another way to manage all the possible conditions. Henceforward, we will consider the notion of *context*. We have decided to relay the evaluation of the context to an entity, called Context Enablement Authority (CEA, see section 4.3 below). The value of the context will be conveyed in the `<Environment>` element as detailed later in this paper.

3.3.3 Subject

The RBAC profile of XACML reduces access control to the concept of roles which may be not sufficient whenever we need to use the user for specific delegation mechanism. For instance, we might want to grant a specific subject, Carol, some special authorizations without creating a specific role for a single subject. According to RBAC, we must consider that she is playing a role by herself (reduced to her subject) and grant this role the selected authorizations. We are not using what XACML is offering, the `subject-id` attribute, which conveys the subject identity.

3.4 Expressing other access control models

The RBAC profile of XACML inherits one of RBAC limitation regarding the abstraction concept. In RBAC, each rule makes use of the role abstraction without taking into consideration other abstractions. These may be the object abstraction and action abstraction. Sometimes multiple objects possess the same security-related properties, that is they should be accessed in the same way. Abstracting them into a class or view avoids the need to write one rule for each of them. This concept has been suggested in the generalized RBAC model (GRBAC) [23], where one considers that through a transducer it is possible to query the system for sets of files that match certain criteria and then consider them to be element of one class. We classify objects into what is called *object role*. The properties that can be taken into consideration in the classification are size, creation date, sensitivity level, ... This concept is also used in the OrBAC model [22], where one defines a view as a set of objects that possesses the same properties within an organization.

Another useful abstraction is that of *action* which is already defined in the OrBAC model. In fact we consider that some actions may somehow fulfill the same operation. For example, the actions `read` (for a file) and `select` (for a database) may be considered as one *consult data* operation. This is why they can be grouped within the same activity for which we may define a single security rule. The existent profile does not take into consideration these other abstractions even though some are defined in the generalized RBAC model. Since XACML is an extensible language where we can add new attributes to the elements defined in the standard, we shall create attributes related to the abstractions of subject, action and object. In this way the extended profile can express the needs of any model based on the idea of views or classes (GRBAC) or also of activities (OrBAC). Note that in this paper we are interested in abstract and expressive access control models that are implementation free. So, we push aside low level access control models like ABAC [27] which is very close to the XACML language but complex to use to express some security rules, in particular, constrained and contextual ones.

4. THE EXTENDED PROFILE

We thought about offering a new profile of XACML that should be a better way to express access control policies within a flexible model that makes the update and manipulation of access rules easier. It is supposed to remove all the limitations stated before.

4.1 The abstract entities

As we have seen before we may need to manipulate abstract entities to manage the access rules more easily. This ability especially useful when updating these rules. We have chosen the following terminology for the abstract entities: *View* is used to classify the objects that share the same properties from a security point of view, *Activity* is the abstract entity of the actions that correspond to the same operation and *Role* is kept as abstraction of subjects.

4.2 The context

One of the limitations of the RBAC profile of XACML is that it cannot express all kinds of conditions of applicability of the rules. One of the main contribution of the OrBAC model in the access control domain is that it can model context that reduces the applicability of the rules to certain circumstances. We have decided to use this definition in our profile to express conditions. In OrBAC we assume that there are five kinds of contexts [17]: the *temporal context*, the *spatial context*, the *user-declared context*, the *prerequisite context* and the *provisional context*. These contexts require some information system that stores and manages different type of information:

- A global clock to check the temporal context. This context depends on the time at which the subject is requesting for an access to the system. Example: the working hours context.
- The subject environment and the software and hardware architecture to check the spatial context. This one depends on the subject location. Example: the secure area context which means that the subject is accessing from the defined secure area such as the intranet in the enterprise.
- The subject purpose to check the user-declared context that rely on the subject objective. Example: the auditing context where the user declares that he is doing an audit and requires other permissions in this specific context.
- A database to check the prerequisite context which depends on the characteristics that join the subject, the action and the object accessed. Example: designated doctor context.
- An history of the actions carried out to check the provisional context that depends on previous actions the subject has performed in the system. Example: limited access context, the access to the resource is limited to a given number of times.

These contexts will be integrated in our profile. As stated before, we admit that there will be a specific entity (called CEA see Section 4.3) that will make the requests to evaluate the value of the activated context(s).

4.3 The Enablement Authorities

We extend the concept of REA defined in the RBAC profile of XACML. We actually define four Enablement Authorities. These authorities are in charge of the translation between the concrete entities and the abstract ones. They include the Role Enablement Authority (REA), the View Enablement Authority (VEA), the Activity Enablement Authority (AEA) and the Context Enablement Authority (CEA). Each of these authorities can interrogate the corresponding assignment policies. We then consider that there is a Role Assignment Policy (RAP) that defines which roles are assigned to a given subject. This policy is used by the REA exactly as defined in the RBAC profile. In the same way we consider a View Assignment Policy (VAP). The VEA can make requests to the VAP to determine to which view(s) the object accessed belongs. The Activity Assignment Policy (AAP) will assign actions into their corresponding activities. The AEA is the authority that manages these assignments via the AAP to get the values of the activities to which belongs the action of the request.

The contextual conditions are taken into consideration by the CEA; it evaluates the context and assigns `<Environment>` element of the XACML policies to the corresponding value. This is why we have defined the Context Assignment Policy (CAP) that states the conditions to activate a given context.

4.4 The policies

The policies written by the PAP are used by the PDP (global access policies) and the Enablement Authorities (assignment policies). These policies use either abstract entities, concrete entities or both. We call abstract (or organizational) policies the policies that use the abstract entities (roles, activities, views) in addition to the context and concrete policies the ones that are expressed using only concrete entities (subject, action and object). Abstract policies may use abstract or concrete entities in addition to the context. In the delegation process we assume that we may need to manipulate a user-user delegation. If we go back to the example where a physician *x*, when she is on holiday, will give permissions to her assistant *y*. Then, the physician *x* may write an organizational policy using the concrete entity (the subject relative to the assistant *y*) giving him permissions to accomplish some activities over some views within a *physician on holiday* context. Now, let us consider the case of partial delegation when the physician *x* need to delegate only a part of her permissions to her assistant *y*. These permissions should be grouped in a new PolicySet B whose Target selects the assistant's identity '*y*' (in a user-user partial delegation) or the role 'assistant' (in a user-role partial delegation). The PolicySet B's target should select also the physician *x*. In this way we maintain the permissions assigned to the physician who will be selected by A and B, whereas the assistant may only be selected by B. The fact that we could express policies using concrete entities may be also seen as a way to grant exceptional permissions to specific users. For instance, one may need to give *Alice* that has the role *surgeon* some specific permissions regardless her role inside the organization. These exceptional permissions, user-user delegations, role-user delegations and role-role delegations are expressed in the assignment policies.

Finally, using the action instead of the activity in an abstract policy may help reducing the permissions accorded to one user to a specific action chosen amongst actions imple-

menting the activity. An example of such a situation is when we define the actions **delete** and **insert** of an object into a table (in a relational database) as belonging to the activity **modify**. Whenever we would like to grant a user the permission to modify the content of the database by inserting an object in a table without allowing him to delete objects from the same table, we may use the action **insert** in the permission instead of the activity **modify**.

Since XACML defines a **deny** effect to a rule, we will use it to express negative authorizations. Then our policies may contain prohibitions in addition to permissions. This implies that we need mechanisms to resolve conflicts that may emerge through the application of multiples rules in a policy, or multiple policies in a PolicySet. These conflicts are solved in the XACML standard by using combining algorithms. While we are confident that techniques exist for conflict resolution (e.g. rule prioritization) [13, 22] and that these techniques are supported by combining algorithms, additional details on this topic are out of the scope of this paper.

4.5 The extended profile architecture

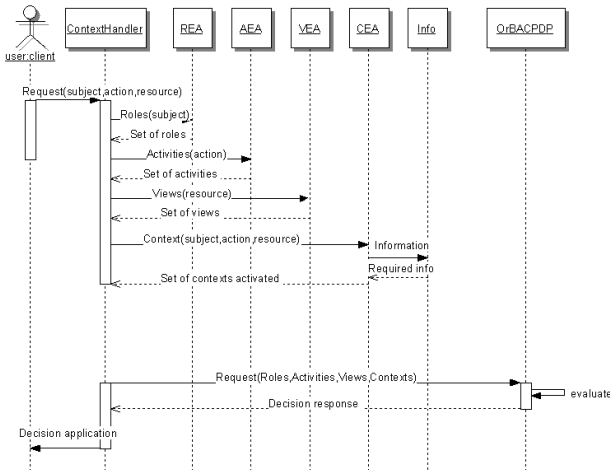


Figure 3: The sequence diagram

Every access request is composed of a user asking if his/her action over a given resource is allowed or not. In the XACML data-flow diagram, the request is taken into consideration by the PEP that will relay it to the context handler. Since the policies are written using abstract or concrete entities, we need to evaluate the abstract entities to which the concrete entities of the request belong but also communicate the concrete values to the PDP for evaluation of exceptional permissions or delegations. This task is achieved by the *Enablement Authorities*. Figure 4 describes the extended profile architecture and figure 3 is the sequence diagram that explains the different steps in this architecture.

The policies are written by the PAP (1). When the PEP receives a concrete request (2) it is relayed to the context handler (3). The context handler will have to do the major part of the work since it is up to it to collect the assignments of each subject to its roles, the resource to its views, the action to its activities and finally get the value of the contexts. This is done by sending requests to the corresponding Enablement Authorities (4). Once all the responses have been

collected (5), the context handler sends a request to the PDP for evaluation (6). This request contains the abstract values and the concrete values of the subject, action and object of the initial request in addition to the value of the context(s). The PDP evaluates the request according to the policies and sends back a response to the context handler (7) that will transmit it to the PEP in its native response language (8). The PEP may have to fulfill some obligations (9) before allowing or denying the access.

After receiving the request from the user, the context handler will send requests to each of the Enablement Authority specifying the necessary attribute values (role, view, activity or context) to be determined. Notice that the evaluation order does not matter. The REA, VEA, AEA and CEA will respectively maintain a list of the defined roles, views, activities and contexts values inside the organization. Each authority will make a request to the corresponding assignment policy for each candidate value. We consider that in order to evaluate certain requests, the CEA may require other information (history of the subject, the date, spatial information, resource content, etc). Then it is the CEA, and not the context handler in our model, that should send additional requests to an information system that manages that kind of information. In this way a part of the normal job of the XACML's context handler, in the evaluation of the request information, is relayed to this new entity. All the values assigned are sent back to the context handler. This latter will add the initial concrete values for each entity and send a request to the PDP for final evaluation.

4.5.1 The hierarchy

Since we suggest abstracting subject, action, object and conditions triggering authorizations, we need to define hierarchies not only between roles but also activities, views and contexts. We will not use the same process to implement the hierarchy as the one defined in the RBAC profile because it is not an intuitive approach. We prefer an assignment process.

Let us consider the role hierarchy. In our solution we will assume that if a subject is assigned to a role r_1 and if this role is a senior role of another role r_2 (junior role) this means that the role r_1 inherits all the authorizations associated with the role r_2 . In this case, we assume that the role r_1 is assigned to the role r_2 as well as the subject will be assigned to both roles r_1 and r_2 . So, we will manage two levels of assignment, the first is a subject assignment to a role and the second is a role assignment to its junior role. These two levels of assignment are managed by the Role Assignment Policy. In this policy we have the global Assignment PolicySet which is made up of two policies: one for the rules related to the subject assignment and the other policy containing the rules related to the role assignment. In this second policy we proceed in the same way as the first one with the only exception that instead of having a subject assigned to a role we will have a role assigned to a role. The other hierarchies are managed in the same way.

5. THE IMPLEMENTATION AND APPLICATION OF THE PROFILE

5.1 The implementation

There are no publicly available implementations of the RBAC profile of XACML. On figure 4, we have drawn a

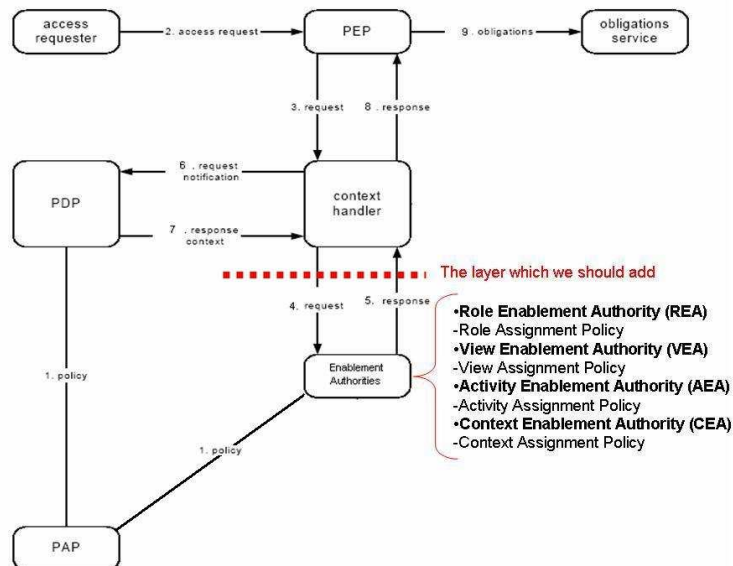


Figure 4: The extended profile architecture

Listing 1: An example to illustrate the new attributes

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
5 http://docs.oasis-open.org/xacml/2.0/
6 access_control-xacml-2.0-context-schema-os.xsd">
7 <Subject>
8 <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
9 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
10 <AttributeValue>urn:example:role-values:physician</AttributeValue>
11 </Attribute>
12 </Subject>
13 <Resource>
14 <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:resource:view"
15 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
16 <AttributeValue>urn:example:view-values:medical_file</AttributeValue>
17 </Attribute>
18 </Resource>
19 <Action>
20 <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:action:activity"
21 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
22 <AttributeValue>urn:example:activity-values:checking</AttributeValue>
23 </Attribute>
24 </Action>
25 <Environment>
26 <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:environment:context"
27 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
28 <AttributeValue>urn:example:environment-values:designated_doctor
29 </AttributeValue>
30 </Attribute>
31 </Environment>
32 </Request>

```

border line separating the architecture defined in the OASIS XACML standard and what we have assumed to be needed to implement our profile. All the work we have done implementation¹ of XACML, version 1.1. We have modified some existing classes and created other ones to implement the Enablement Authorities. The classes included in the available implementation allowed us to create a PDP that is able to evaluate any XACML request considering an XACML policy. This is a fundamental function we need in our architecture. Actually, each Enablement Authority has somehow the same capabilities as a PDP since it should

¹<http://sunxacml.sourceforge.net/>

evaluate, according to the available assignment policies, the request sent by the context handler.

Our profile does not depend on the implementation choices of the application in hand. In other words, the way the policies and the abstract values can be made available to the Enablement Authorities is not imposed. This can be done using an LDAP server, a database or a file system. In our implementation, we have extended the XACML language with new attributes to manage the abstract entities that we have considered (role, view and activity) and context. New attributes have been defined to handle the role values (XACML subject attributes), the view values (XACML resource attributes), the activity values (XACML action attributes) and finally the context values (XACML environment attributes).

Listing 1 shows an illustration of these extensions. It is an example of an XACML request the context handler may send to the PDP for evaluation. On lines 8, 14, 20 and 26 we can see the identifier attribute of our new attributes. We use two levels of assignment in the assignment policies to implement the role, activity, view and context hierarchies. (1) Each Enablement Authority evaluates the query in hand to determine the value of the entity it has in charge (role value, activity value, view value, context value) that matches the query entities (subject, action, object). The response of the evaluation can be permit, deny, NotApplicable or Indeterminate for each request. The response permit means that the corresponding value is assigned. (2) After receiving the assignment values (those for which the XACML response is permit), the Enablement Authorities will reiterate the evaluation for each value found to be sure that there are no additional junior values assigned.

Listing 2 illustrates an example of an action/activity assignment. In this example, a rule will respond to a request sent by the AEA asking whether a subject, the action read (line 9), can perform the action enableActivity (line 25) over the resource the activity checking (line 17). In other words,

Listing 2: An example of an activity assignment policy

```
1 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicyId="Activity:Assignment:Policy"
2 RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
3 <Target/>
4 <Rule RuleId="checking:activity:requirements" Effect="Permit">
5 <Target>
6 <Subjects>
7 <Subject>
8 <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
9 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
10 <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
11 </SubjectMatch>
12 </Subject>
13 </Subjects>
14 <Resources>
15 <Resource>
16 <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
17 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">urn:example:activity-values:checking</AttributeValue>
18 <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:action:activity" DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
19 </ResourceMatch>
20 </Resource>
21 </Resources>
22 <Actions>
23 <Action>
24 <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
25 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">urn:oasis:names:tc:xacml:2.0:actions:enableActivity</AttributeValue>
26 <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
27 </ActionMatch>
28 </Action>
29 </Actions>
30 </Target>
31 </Rule>
32 </Policy>
```

this rule states that it is permitted to assign the action `read` to the activity `checking`. This is the first level of assignment (see Section 4.5.1) whereas in the second level of assignment, the action `read` should be replaced by an activity value. For instance, we may have another rule stating that it is permitted to assign the action `checking`, which is in this case an activity value, to the activity `consulting`. This is a second level assignment rule.

5.2 The applicability of the profile

The extensions we suggest and implement does not inhibit the functionalities of the existent RBAC profile. It is obvious that our extended profile supports policies expressed using only the concept of roles. The RBAC hierarchy can still be expressed, in the extended profile, using the role assignment `PolicySet` rather than the `Permission PolicySet`. Moreover, the proposed profile has added new and useful concepts to resolve some of the problems of the RBAC profile like delegation and sophisticated contextual authorizations (see Section 4.4).

Our extended profile will also integrate other models than the RBAC model. It can handle the security requirements of both very basic access control models (like DAC) and more expressive ones (like GRBAC, TBAC or OrBAC).

We have seen that entities' abstractions allow us to better express the security requirements of the GRBAC model. GRBAC introduces subject roles and object roles that are managed in the extended profile by the "role" and "view" entities. Moreover, GRBAC model incorporates environment roles that capture environmental information, such as time of day or system load. These roles are expressed by the "context" notion managed within our profile. TBAC model can easily be expressed in the extended profile as it is based on the abstraction of actions into tasks. These tasks are mapped to the 'activity' entities in our profile. Furthermore, we can integrate the OrBAC model, an expressive model based on the concept of organization. In this model, both the *concrete entities* (*subject*, *action* and *object*) and their abstractions, the *organizational entities* (*Role*, *View* and *Activity*), are used. This model defines also a new en-

tity, the *context*. These concrete and abstract entities are all expressed in our profile. OrBAC defines hierarchies of roles, views, activities and contexts. It not only defines a role-role delegation but also a role-user or user-user delegation. These hierarchies and delegations are handled by our profile. Finally, like the OrBAC model, we can specify negative authorizations in the extended profile. So, an OrBAC security policy is easily transcribed in the extended profile we propose.

Many existing works also attempt to extend the RBAC model to deal with contextual conditions such as the users' location context (for instance the GEO-RBAC model [14]) or temporal context (for instance the GTRBAC model [21]). These approaches suggest combining the concept of role with spatial or temporal contextual conditions to obtain "contextual" roles. In this paper, we follow a different approach since a context is not attached to the role but to the security rules. We actually argue that the notion of "contextual" role generally corresponds to artificial roles and sometimes to ambiguous roles. For instance, if *physician* is a role and *H1* is a hospital, a "combined" role such as *H1_physician* is ambiguous since it may be interpreted as a "contextual" role (corresponding to *physician located* in hospital *H1*) or an "organizational" role (corresponding to *physician empowered* in hospital *H1*).

6. CONCLUSION AND PERSPECTIVES

We have detailed in this paper the use of the XACML as an access control language. We have seen that it is very helpful to link it with some existent access control model. This will help a security manager to model all the security requirements before specifying them using a standard right expression language. The existent RBAC profile of XACML, that explains how the XACML does meet the RBAC model requirements, has shown some limitations. It does not include all the functionalities an access control may require. Furthermore, it reduces the capabilities, and does not make use of all the advantages, of the XACML language that is considered to be a very powerful and rich language.

We have proposed to extend the RBAC profile of XACML with new functionalities that should overcome its current limitations. We have shown that the proposed profile is general enough to express, in addition to RBAC, other access control models such as GRBAC, TBAC and OrBAC.

We have used the same process of role assignment, proposed in the RBAC profile of XACML, in the other assignment processes (activity, view and context). Then we have retained one of the primary problems with the RBAC profile: it requires prior knowledge of potential roles, views, activities and contexts. Each of them should be tested individually for each request. This appears to be expensive. Performance should be studied in the future and some improvement should be done in the way each request is evaluated.

We are planning to take into consideration the issue of policy administration [9] within this profile. Furthermore, we are aiming to analyze this profile in terms of interoperability which is one of the major topics in the domain of web services [2, 1].

7. REFERENCES

- [1] Web-services policy language use-cases and requirements. Working draft 04, OASIS, April 2003.
- [2] XACML profile for Web-services. Working draft 04, OASIS, September 2003.
- [3] Core and hierarchical role based access control (RBAC) profile of XACML v2.0. Standard, OASIS, February 2005.
- [4] eXtensible Access Control Markup Language (XACML) Version 2. Standard, OASIS, February 2005.
- [5] SAML 2.0 profile of XACML v2.0. Standard, OASIS, February 2005.
- [6] SAML V2.0 Executive Overview. Committee DRAFT 01, OASIS, April 2005.
- [7] Security Assertion Markup Language (SAML) V2.0 Technical Overview. Working Draft 07, OASIS, July 2005.
- [8] Web Services Security: Key Industry Standards and Emerging Specifications Used for Securing Web Services. White Paper, Computer Associates, January 2005.
- [9] XACML v3.0 administrative policy. Working draft 10, OASIS, December 2005.
- [10] Web Services Security : SOAP Message Security 1.1(WSSecurity 2004). Standard Specification, OASIS, February 2006.
- [11] Yuan Rao and Bo-Qin Feng and Jin-Cang Han and Zun-Chao Li. SX-RSRPM: a security integrated model for Web services. In *Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai*, August 2004.
- [12] Ezedin S. Barka and Ravi S. Sandhu. Framework for Role-Based Delegation Models. In *16th Annual Computer Security Applications Conference, New Orleans, Louisiana*, December 2000.
- [13] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A Logical Framework for Reasoning about Access Control Models. *ACM Transactions on Information and System Security*, 6(1), February 2003.
- [14] E. Bertino, E. Catania, M. Damiani, and P. Persasca. GEO-RBAC: A Spatially Aware RBAC. In *10th ACM Symposium on Access Control Models and Technologies (SACMAT 2005)*, Chantilly, Virginia, USA, May 2001.
- [15] L. Cholvy and F. Cuppens. Analyzing Consistency of Security Policies. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.
- [16] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and Systems Security (TISSEC)*, 4(3), 2001.
- [17] Frédéric Cuppens and Alexandre Miège. Modelling Contexts in the Or-BAC Model. *ACSAC*, page 416, 2003.
- [18] George Yee and Larry Korba. Negotiated Security Policies for E-Services and Web Services. In *Proceedings of the 2005 IEEE International Conference on Web Services (ICWS)*, July 2005.
- [19] M.A. Harrison, M.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.
- [20] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible Support for Multiple Access Control Policies. *ACM Transactions on Database Systems*, 26(2), June 2001.
- [21] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. Generalized Temporal Role-Based Access Control Model. 17(1):4–23, January 2005.
- [22] Alexandre Miège. *Définition of a formal framework for specifying security policies. The Or-BAC model and extensions*. PhD thesis, ENST, June 2005.
- [23] M.J. Moyer and M. Abamad. Generalized role-based access control. *21st International Conference on Distributed Computing Systems*, pages 391–398, April 2001.
- [24] R Sandhu, D Ferraiolo, and R Kuhn. The NIST model for role-based access control: towards a unified standard. *Proceedings of the fifth ACM workshop on Role-based access control*, pages 47–63, 2000.
- [25] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, February 1996.
- [26] R. K. Thomas and R. S. Sandhu. Task-based Authorization Controls(TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management. In *Proceedings of the IFIP WG11.3 Workshop on Database Security, Lake Tahoe, California*, August 1997.
- [27] E. Yuan and J. Tong. Attribute based access control (ABAC): a new access control approach for service oriented architectures. *Ottawa New Challenges for Access Control Workshop*, April 2005.